Performance-Efficiency Trade-off of Low-Precision Numerical Formats in Deep Neural Networks

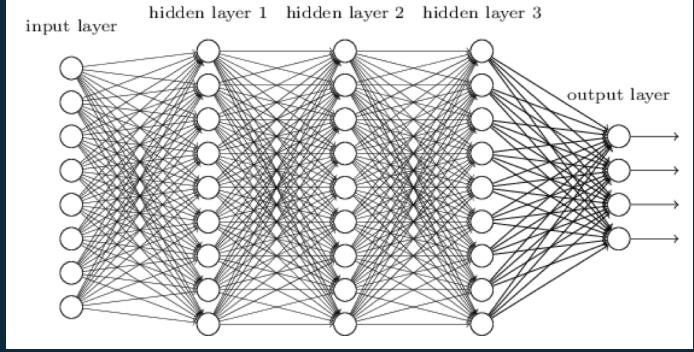
Authors: Zachariah Carmichael, Hamed F. Langroudi, Char Khazanov, Jeffrey Lillie, John L. Gustafson, Dhireesha Kudithipudi
Neuromorphic AI Lab, Rochester Institute of Technology, NY, USA

RIT Rochester Institute of Technology

Background

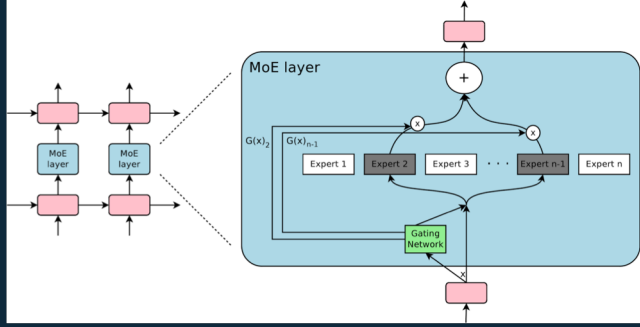
- Deep Neural Networks
- AI on Edge-Devices
- Posits

Deep Neural Networks (DNNs)



[14]

- Highly-parallel connectionist networks
- DNNs learn a nonlinear input-to-output function by minimizing an objective via gradient descent and backpropagation with labeled data (in general)
- Successfully applied to various domains: natural language processing, genomics, computer vision

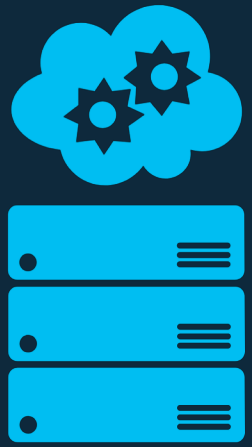


Mixture-of-Experts network with >137 billion parameters (~548 GB memory with 32-bit floats) [2].

DNNs are Hungry

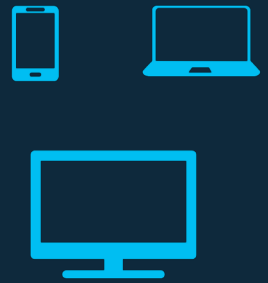
- Multiply-and-Accumulate (MAC) units ubiquitous across DNNs
- MAC operations consume compute resources
- Data & DNN parameters consume memory

Development



Training
Evaluation
Tuning

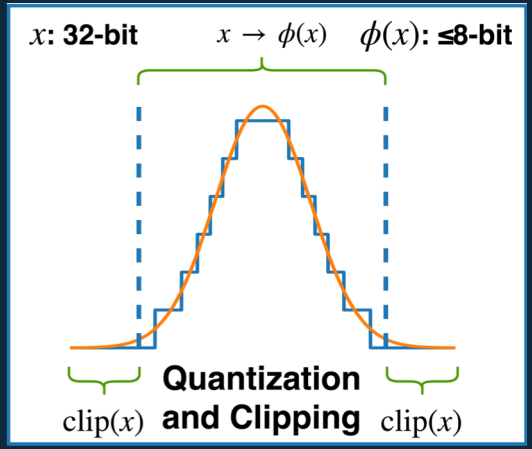
Deployment



Inference

How can DNNs be deployed **tractably** on edge devices without sacrificing **performance**?

Quantization



- Linear & Nonlinear Quantization
  - On-device compute overhead
  - May require retraining
  - May require an increase in DNN parameters or hyperparameters [3]

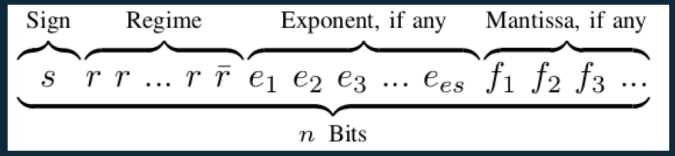
- **Direct Quantization**
  - No compute overhead
  - Relies on natural distribution of numerical format for maintaining inference performance

There is a need for a **fair** comparison between **low-precision** numerical formats.

How do the natural distributions of numerical formats compare during low-precision DNN inference?

- Fixed-point
- Floating point
- **Posit**

Posit Number System

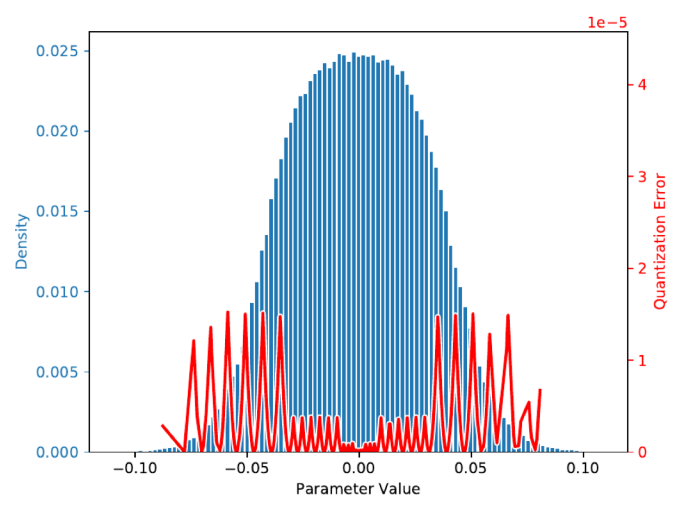


Bitwise interpretation of a posit number

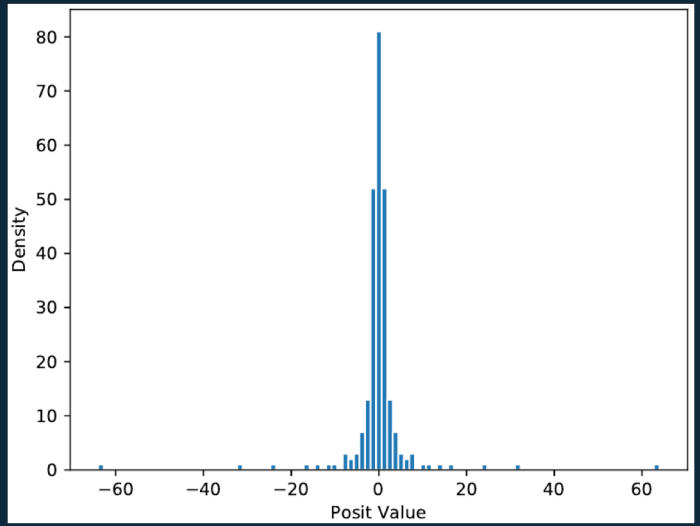
$$(-1)^s \times \left(2^{2^{es}}\right)^k \times 2^e \times 1.f$$

- Proposed as improvement over IEEE-754 floating point [5]
  - Better dynamic range
  - Higher accuracy
  - Clean: no redundant representations
  - No subnormals
  - Guaranteed program reproducibility
- Tapered-precision
- es parameter controls the dynamic range

Intuition for Posits



ConvNet weight distribution and quantization error for 8-bit posit, es=0.



8-bit posit, es=0, value distribution.

Methodology

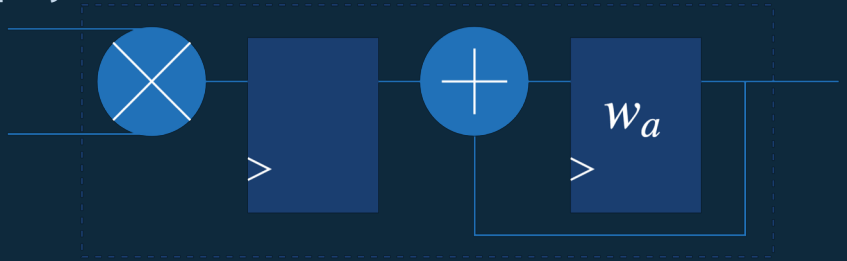
- The Exact MAC Unit
- The DNN Accelerator

$$w_a = \lceil \log_2(k) \rceil + 2 \times \left\lceil \log_2 \left(\frac{max}{min} \right) \right\rceil + 2$$

Fixed-point accumulator width. *max* and *min* are the maximum and minimum absolute values of a number format, respectively.

The Exact MAC (EMAC) Unit

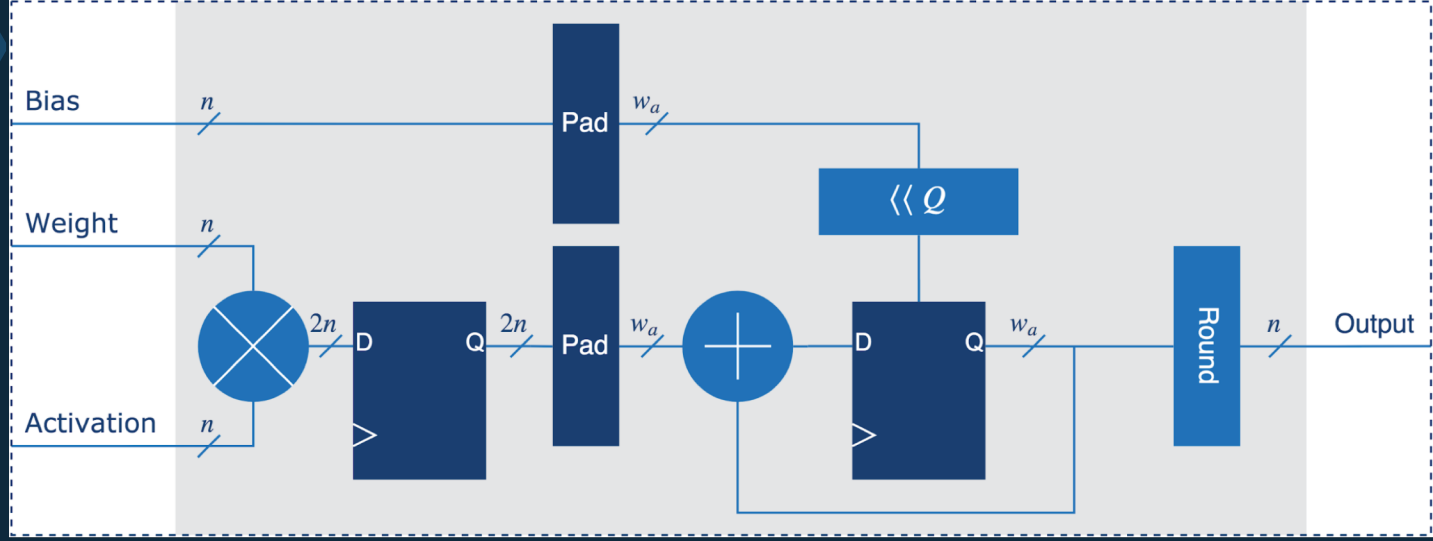
- Extend the MAC to the EMAC
- Wide accumulator architecture remains resource-efficient at low-precision
  - Accumulate in fixed-point
- Rounding & truncation delayed until sum of *k* products is computed
- Employed for all numerical formats considered



The Kulisch Exact Dot Product

- Exact operations become vastly more impactful at low-precision
- Kulisch [7]:
  - Compute exact sums of products
  - A wide register accumulates fixed-point values shifted by an exponential parameter, if applicable
  - No rounding or truncation until result needed

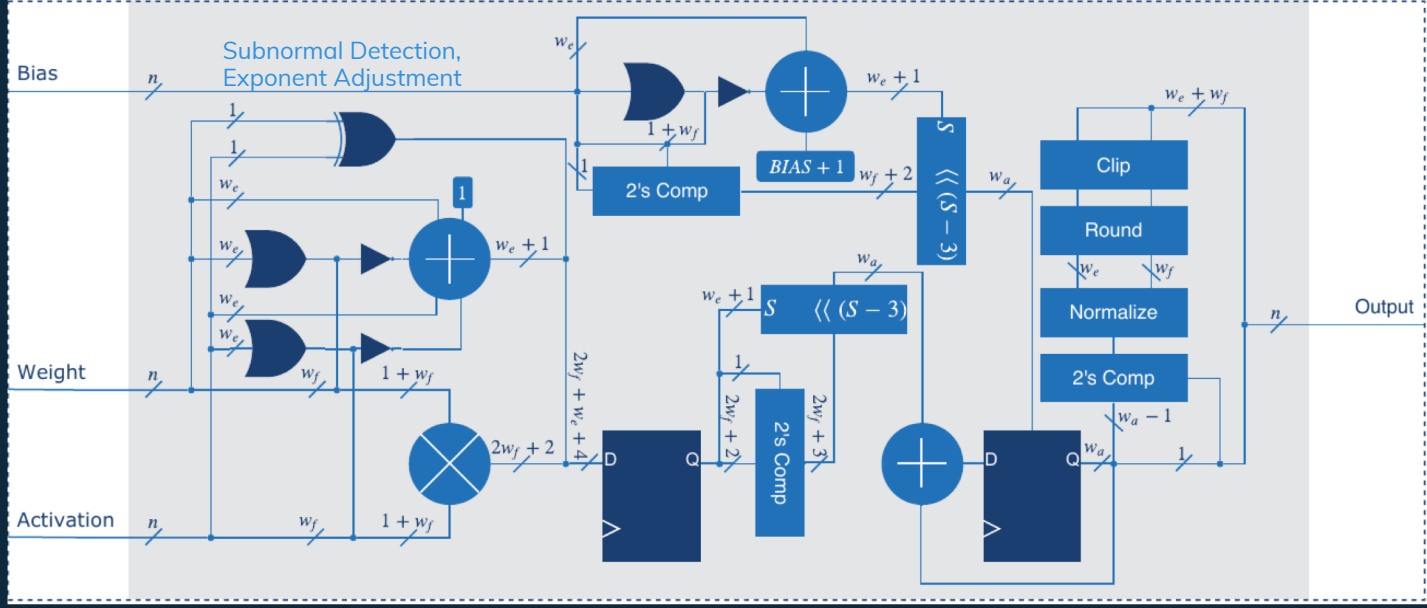
Fixed-Point



The fixed-point parametrized EMAC.

- Parametrized as *n*-bit number with Q fractional bits
- All EMACs receive a weight, bias, and activation/input data feature
- Rounding occurs after accumulation of the *k* products
  - Round-to-nearest, ties to even
- The wide accumulator is reset to the “Bias” at the beginning of each dot product
- No underflow/overflow, values clipped to *max*

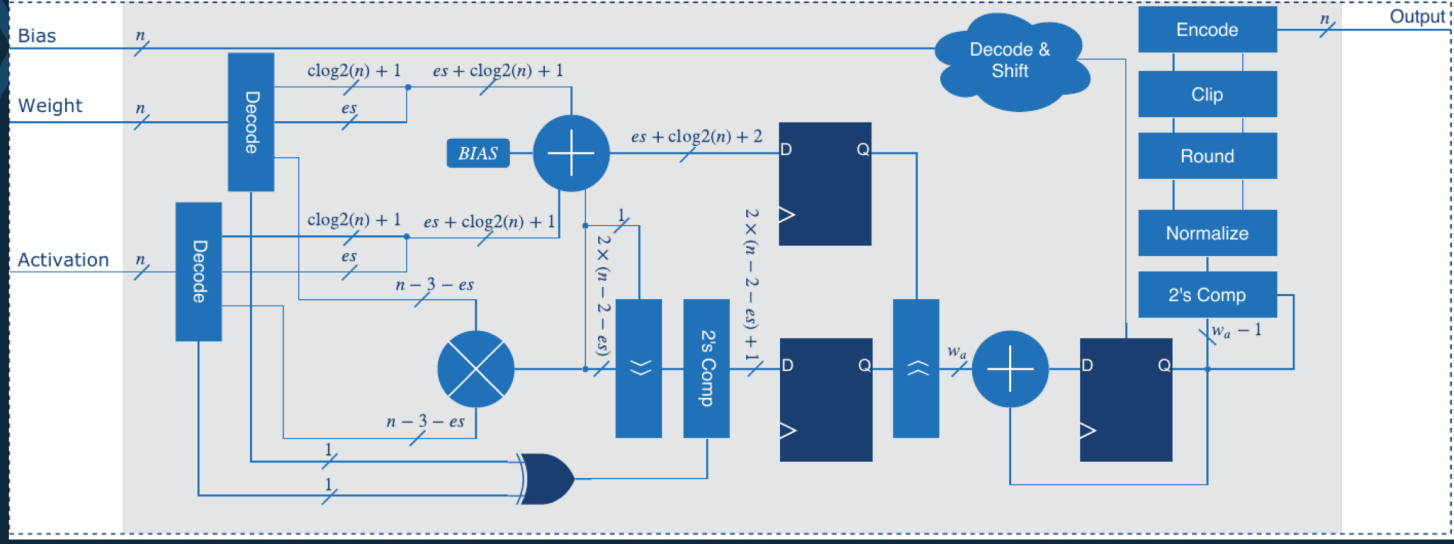
Floating Point



The floating point parametrized EMAC.

- Parametrized as *n*-bit number with w_e exponential bits and w_f fractional bits
- NaN, +/- Inf not considered in implementation
- Subnormal detection required before multiplication and during exponent/mantissa packing
- The biased exponent is used to shift the product into its fixed-point representation for accumulation
- Single leading-zeros-detector (LZD) used to encode exponent and extract mantissa

Posit

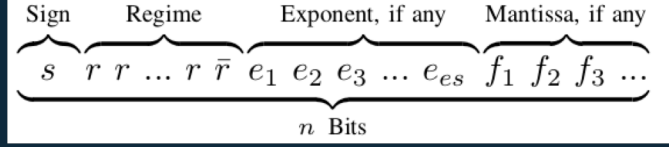


The posit parametrized EMAC.

- Parametrized as *n*-bit number with es exponent bits
- NaR (Not-a-Real) not considered in implementation
- The biased exponent is used to shift the product into its fixed-point representation for accumulation
- Single leading-zeros-detector (LZD) used to encode exponent and extract fraction bits

Algorithm 1 Posit data extraction of n-bit input with es exponent bits

1: **procedure** DECODE(in) ▷ Data extraction of in
2: nzero $\leftarrow$ |in ▷ '1' if in is nonzero
3: sign $\leftarrow$ in$[n-1]$ ▷ Extract sign
4: twos $\leftarrow$ $(\{n-1\{\text{sign}\}\} \oplus \text{in}[n-2:0]) + \text{sign}$ ▷ 2's Comp.
5: rc $\leftarrow$ twos$[n-2]$ ▷ Regime check
6: inv $\leftarrow$ $\{n-1\{\text{rc}\}\} \oplus$ twos ▷ Invert 2's
7: zc $\leftarrow$ LZD(inv) ▷ Count leading zeros
8: tmp $\leftarrow$ twos$[n-4:0] \ll (\text{zc}-1)$ ▷ Shift out regime
9: frac $\leftarrow$ $\{\text{nzero}, \text{tmp}[n-es-4:0]\}$ ▷ Extract fraction
10: exp $\leftarrow$ tmp$[n-4:n-es-3]$ ▷ Extract exponent
11: reg $\leftarrow$ rc ? zc-1 : $-$zc ▷ Select regime
12: **return** sign, reg, exp, frac
13: **end procedure**



$$k = \begin{cases} n\_reg - 1 & , \text{ if } r = 1 \\ -n\_reg & , \text{ if } r = 0 \end{cases}$$

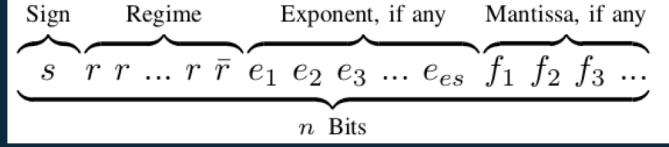
Posit Decoding

- Regime and exponent can be extracted naturally
- Shifting posit by *n_reg regime* bits gives e and *f*
  - Single LZD is used to extract *n_reg*
- As e is es bits, the scale factor is given simply by concatenating *k* and e

$$(-1)^s \times \left(2^{2^{es}}\right)^k \times 2^e \times 1.f$$

$$\left(2^{2^{es}}\right)^k \times 2^e = 2^{(2^{es} \times k)} \times 2^e = 2^{(2^{es} \times k + e)}$$

Scale factor



Posit Encoding

- Similarly, *k* and e can be taken easily from the scale factor
- Absolute value of *k* shifts e and *f* into place
  - Positive *k*: arithmetic shift right by *k*
  - Negative *k*: shift right by abs(*k*)-1
- Example:

Example (*es* = 1):

scalefactor = 11_{10} = 01011_2

→ *k* = 5_{10} = 0101_2

→ *e* = 1_{10} = 1_2

Verify:

2^{es} x 5 + 1 = 11

k = -2 (es = 1)

01*efffff|fff* → 001*efff|fff*

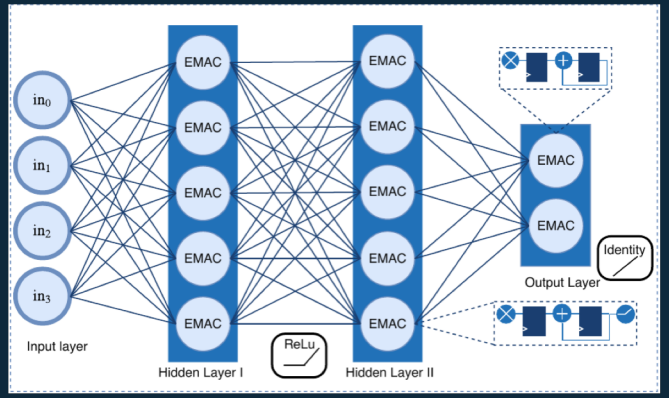
k = 2 (es = 1)

10*efffff|fff* → 1110*eff|fff*

$$2^{es} \times k + e$$

Scale factor

Deep Positron Accelerator



[1]

- Simple FPGA-accelerated fully-connected feedforward neural network
  - Generic network size (layers & hidden units)
  - Parameterized EMAC number format (fixed-point, floating point, posit)
- Local memory per EMAC for DNN parameters
- ReLU activations, linear readout layer
- Inference only
- Xilinx Virtex-7 FPGA (xc7vx485t-2ffg1761c)

Results

- Supervised Tasks
- EMAC Comparisons
- Performance-Efficiency Trade-offs

The Tasks

- Wisconsin (WI) Breast Cancer [8]:
  - Malignant/Benign classification from 10 cellular features
- Iris [9]:
  - Iris flower ternary classification from 4 sepal/petal measurements
- Mushroom [10]:
  - Mushroom edibility (binary) classification from 22 categorical attributes
- MNIST [11]:
  - Handwritten digit (0-9) classification (784-pixel images)
- Fashion MNIST [12]:
  - Drop-in replacement for MNIST for clothing classification (tougher than MNIST)

[13]

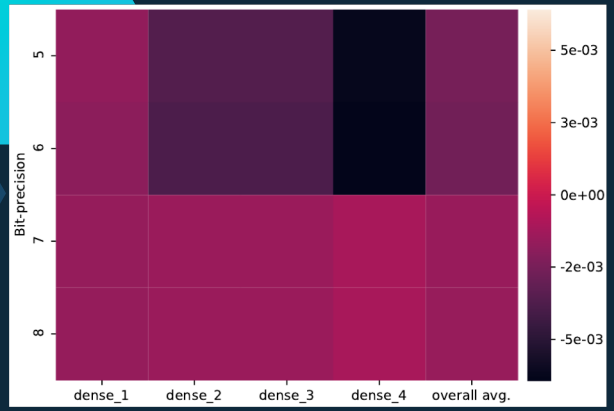

[12]

DNN Training & Inference

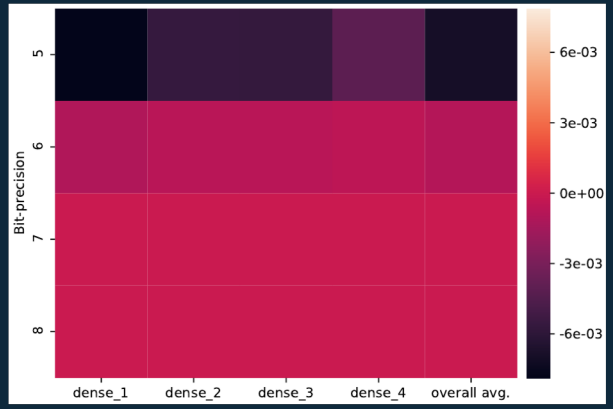
- High-precision training, low-precision inference
- DNNs trained with floating point at 32-bit precision
- Direct Quantization to [5,8]-bit {fixed,float,posit} for inference
  - Round-to-nearest with ties to even
- Format parameters are swept during inference
  - Posit: es
  - Floating point: w_e
  - Fixed-point: Q

Quantization Error

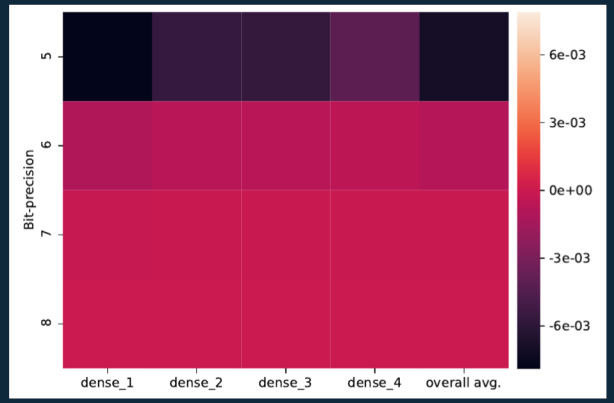
$$\mathrm{MSE}(X, X_{quant}) = \frac{1}{n}\sum_{i}^{n}(X - X_{quant})^2$$



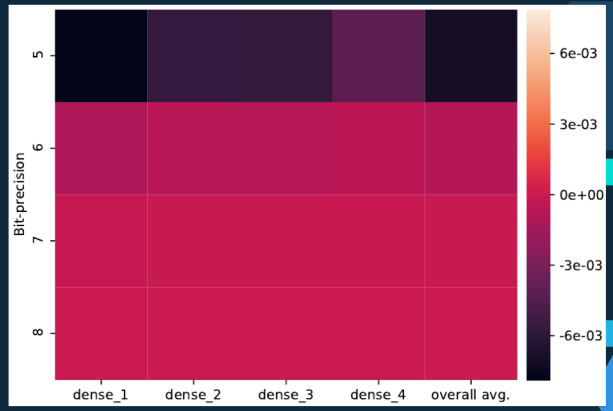
MNIST MSE_{posit} - MSE_{fixed}



Fashion MNIST MSE_{posit} - MSE_{fixed}



MNIST MSE_{posit} - MSE_{float}



Fashion MNIST MSE_{posit} - MSE_{float}

Task Results (8-bit)

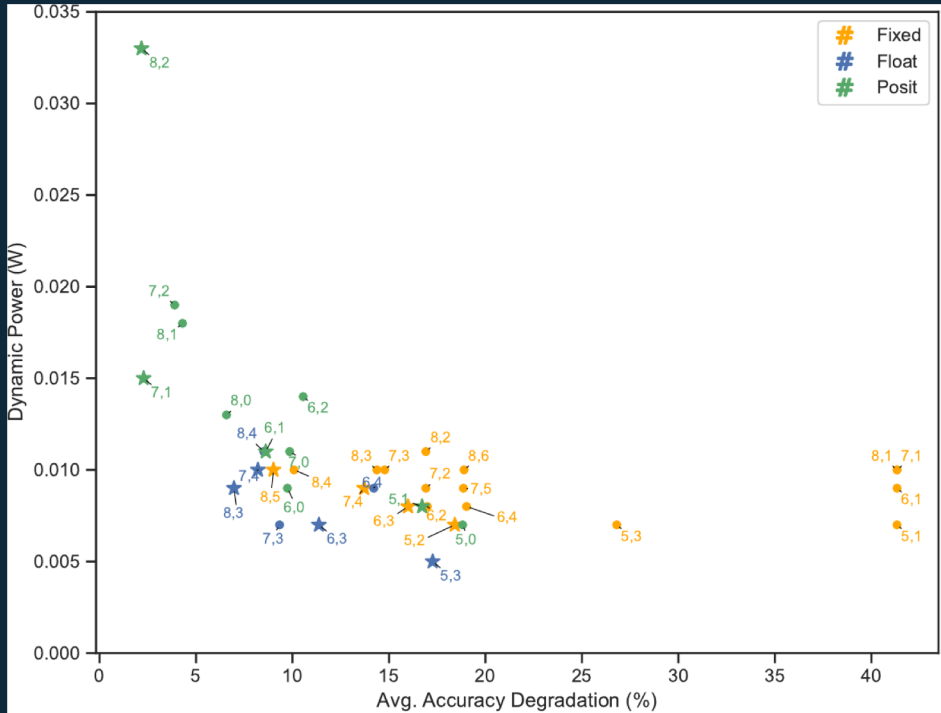
Dataset	Inference Size	Posit	Floating Point	Fixed-Point	32-bit Float
		Acc. (es)	Acc. (w_e)	Acc. (Q)	Acc.
WI Breast Cancer	190	**85.9**% (2)	77.4% (4)	57.8% (5)	90.1%
Iris	50	**98.0**% (1)	96.0% (3)	92.0% (4)	98.0%
Mushroom	2,708	**96.4**% (1)	**96.4**% (4)	95.9% (5)	96.8%
MNIST	10,000	**98.5**% (1)	98.4% (4)	98.3% (5)	98.5%
Fashion MNIST	10,000	**89.6**% (1)	**89.6**% (4)	89.2% (4)	89.5%

- Posit outperforms on all 5 tasks with direct quantization
  - Same accuracy as 32-bit floats in some cases

Trade-off: EMAC Power

⋆: Lowest accuracy degradation (from 32-bit floating point)

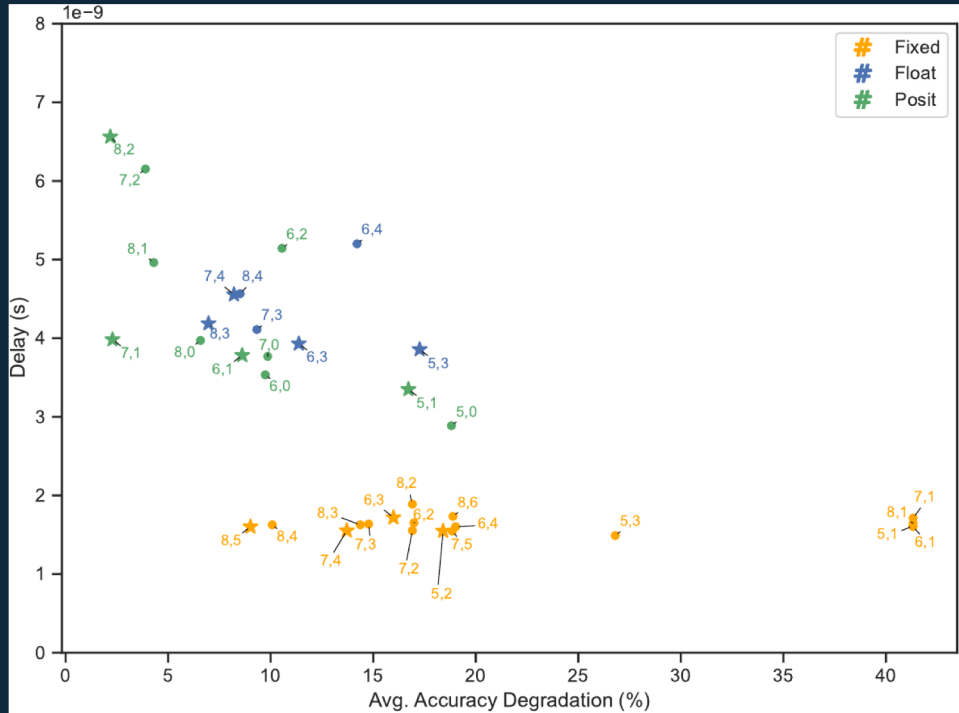
- Evaluated for [5,8] bit widths
- Floating point can be realized with a lower dynamic power requirement than posits
  - Also lower than fixed-point at <7 bits (in several cases)
- Posit at es=0 has competitive power consumption with floats
  - Consumes exponentially more power as es (dynamic range) grows



Trade-off: EMAC Latency

⋆: Lowest accuracy degradation (from 32-bit floating point)

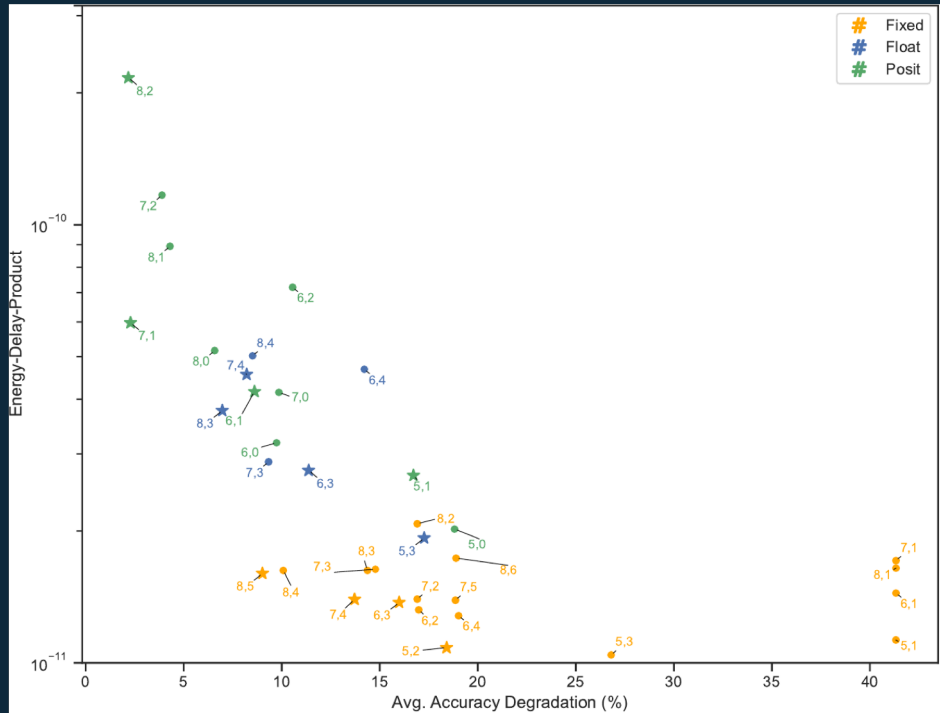
- Fixed-point (obviously) exhibits lowest latency, but worst performance
- Posits outperform floating point in both bit-width and latency at all bit-widths ([5,8])
- Posit latency increases rapidly as es increases
  - However, es=0/1 still outperforms in both latency and degradation compared to floats



Trade-off: EMAC Energy-Delay-Product

- Fixed-point exhibits lowest EDP, but worst performance
- Posits have competitive EDP with floats at similar accuracy degradations
  - Floats generally have lower EDP
- Deep Positron with posit (es = 1) has better energy-delay-product and accuracy for [5, 7] bits
  - For 8-bit, es = 1 is a better fit for energy-efficient applications and es = 2 for accuracy-dependent applications

⋆: Lowest accuracy degradation (from 32-bit floating point)



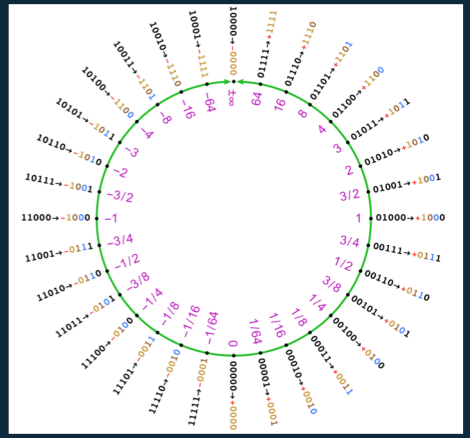
Comparison of posit arithmetic hardware implementations

Design	[17]	[3]	[25]	[4]	[23]	[18]	This Work
Device	Virtex-6 FPGA/ASIC	Zynq-7000 SoC/ASIC	Stratix V GX 5SGXA7 FPGA	Virtex7 VX690 & Ultrascale Plus VU3P FPGAs	Artix-7 FPGA	ASIC	Virtex-7 (xc7vx485t-2ffg1761c) FPGA
Task	-	FIR Filter	-	-	-	Image Classification	Image Classification
Dataset	-	-	-	-	-	ImageNet	WI Breast Cancer, Iris, Mush-room, MNIST, Fashion MNIST
Bit-precision	All	All	All	32	All	All, emphasized on 8	All, emphasized on [5, 8]
Operations	Mul,Add/Sub	Mul,Add/Sub	Mul,Add/Sub	Quire	Quire	Quire	Quire
Programming Language	Verilog	Verilog	C++ /OpenCL	Verilog	C#	OpenCl	VHDL
Technology Node	40 nm / 90 nm	28 nm / 90 nm	28 nm	28 nm / 20 nm	28 nm	28 nm	28 nm

Conclusion

- Posits are a natural fit for low-precision DNN inference
- Posits can be fine-tuned for either accuracy-critical or latency-critical applications

- A system level optimization is required for large scale analysis and energy savings(with EMACs)



[6]

References

[1] Carmichael, Z. et al. 2019. Deep Positron: A Deep Neural Network Using Posit Number System. In Design, Automation & Test in Europe (DATE) Conference & Exhibition. IEEE.

[2] Shazeer, N. et al. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. arXiv preprint arXiv:1701.06538(2017).

[3] Mishra, A. and Marr, D. 2018. WRPN & Apprentice: Methods for Training and Inference using Low-Precision Numerics. arXiv preprint arXiv:1803.00227(2018).

[4] Carmichael, Z. et al. 2019. Performance-Efficiency Trade-off of Low-Precision Numerical Formats in Deep Neural Networks. In Conference for Next Generation Arithmetic 2019 (CoNGA'19). ACM.

[5] Gustafson, J.L. and Yonemoto, I.T. 2017. Beating Floating Point at its Own Game:Posit Arithmetic.Supercomputing Frontiers and Innovations 4, 2 (2017), 71–86.

[6] J. Gustafson, "Next Generation Arithmetic for HPC and AI: An Update", Rochester Institute of Technology Graduate Seminar, 2018. Used with permission.

[7] Kulisch, U. 2013.Computer arithmetic and validity: theory, implementation, and applications. Vol. 33. Walter de Gruyter.

[8] Street, W.N. et al. 1993. Nuclear feature extraction for breast tumor diagnosis. In Biomedical Image Processing and Biomedical Visualization, Vol. 1905. International Society for Optics and Photonics, 861–871.

[9] Fisher, R.A. 1936. The use of multiple measurements in taxonomic problems.Annals of eugenics 7, 2 (1936), 179–188.

[10] Schlimmer, J.C. 1987. Concept acquisition through representational adjustment.(1987)

[11] LeCun, Y. 1998. The MNIST database of handwritten digits. http://yann. lecun.com/exdb/mnist/(1998)

[12] Xiao, H. et al. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.arXiv preprint arXiv:1708.07747(2017).

References

[13] Medium. (2019). Iris Flower Classification – Srishti Sawla – Medium. [online] Available at: https://medium.com/@srishtisawla/iris-flower-classification-fb6189de3fff.

[14] Nielsen, M. (2019). Neural Networks and Deep Learning. [online] Neuralnetworksanddeeplearning.com. Available at: http://neuralnetworksanddeeplearning.com/chap6.html.

[18] Johnson, J. 2018. Rethinking floating point for deep learning. arXiv preprint, arXiv:1811.01721 (2018).